Efficiency and Accuracy Issues for Sampling vs. Counting Modes of Performance Monitoring Hardware

Shirley Moore,

University of Tennessee-Knoxville

Patricia Teller and Michael Maxwell,
University of Texas-El Paso

Outline

- Background
- PAPI
- Examples of counting and sampling modes
- Accuracy Issues
- Overhead Issues
- Implications for PAPI
- Microbenchmark results and future work

Performance Monitoring Hardware

- Available on most modern microprocessors
- Consists of registers that record data about the processor's function
  - Event counts
  - Data and instruction addresses for an event
  - Pipeline or memory latencies
- Control registers for configuration and control
- Data useful for performance modeling, analysis, and tuning

Overview of **PAPI**

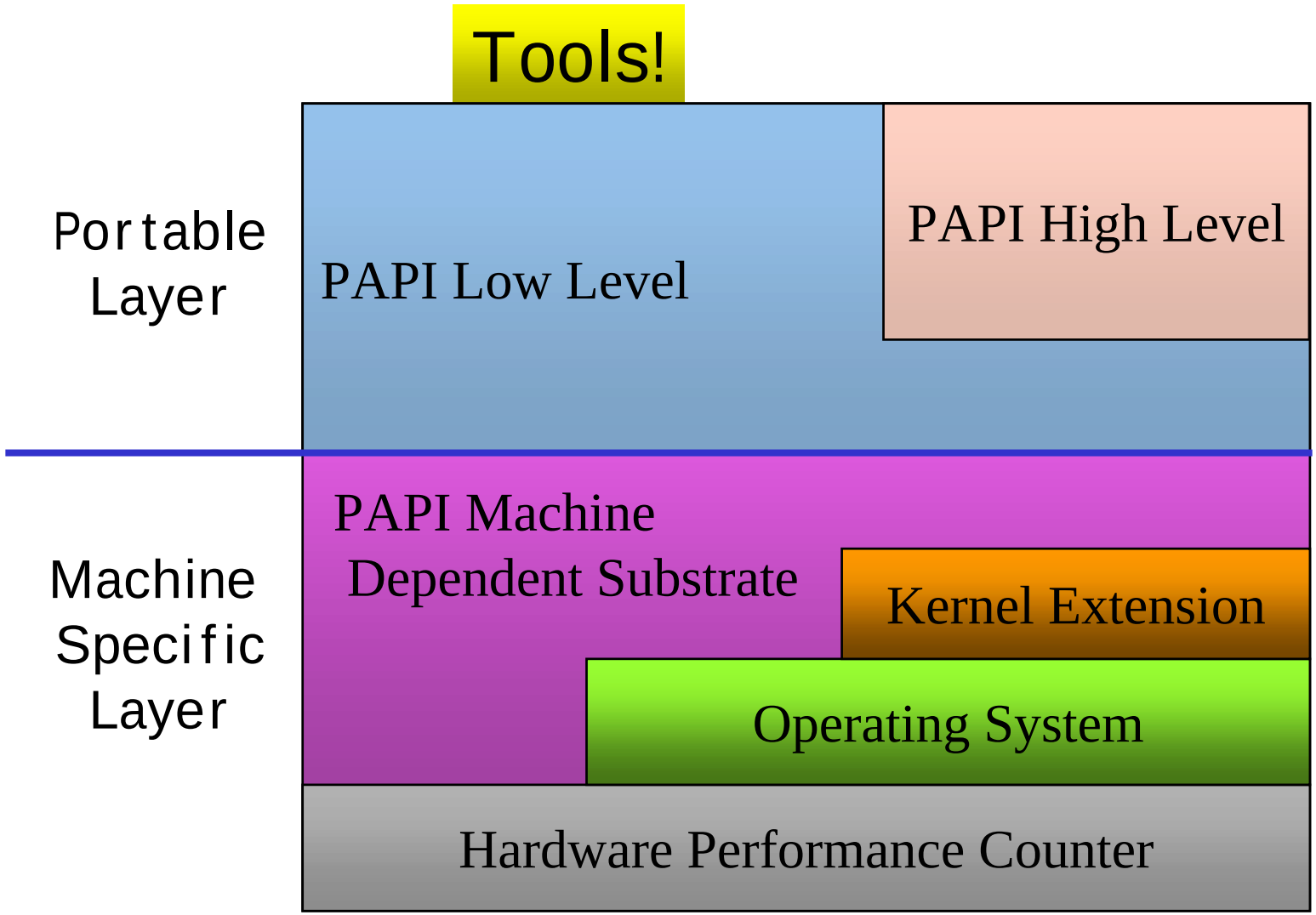
- **P**erformance **A**pplication **P**rogramming **I**nterface
- The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- Parallel Tools Consortium project

  http://www.ptools.org/

PAPI Counter Interfaces

- PAPI provides three interfaces to the underlying counter hardware:
  1. The low level interface manages hardware events in user defined groups called *EventSets*.
  2. The high level interface simply provides the ability to start, stop and read the counters for a specified list of events.
  3. Graphical tools to visualize information.

PAPI Implementation



PAPI Preset Events

- Proposed standard set of events deemed most relevant for application performance tuning
- Defined in papiStdEventDefs.h
- Mapped to native events on a given platform
  - Run tests/avail to see list of PAPI preset events available on a platform

High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Not thread safe
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (additional code) than low-level

High-level API

- C interface
  PAPI_start_counters
  PAPI_read_counters
  PAPI_stop_counters
  PAPI_accum_counters
  PAPI_num_counters
  PAPI_flops
- Fortran interface
  PAPIF_start_counters
  PAPIF_read_counters
  PAPIF_stop_counters
  PAPIF_accum_counters
  PAPIF_num_counters
  PAPIF_flops

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- Manages events in EventSets
- About 40 functions
- Obtain information about the executable and the hardware
- Thread-safe
- Fully programmable
- Callbacks on counter overflow
- Counter multiplexing

Event set Operations

- Event set management
  PAPI_create_eventset, PAPI_add_event[s], PAPI_rem_event[s], PAPI_destroy_eventset
- Event set control
  PAPI_start, PAPI_stop, PAPI_read, PAPI_accum
- Event set inquiry
  PAPI_query_event, PAPI_list_events,...

Callbacks on Counter Overflow

- PAPI provides the ability to call user-defined handlers when a specified event exceeds a specified threshold.
- For systems that do not support counter overflow at the OS level, PAPI sets up a high resolution interval timer and installs a timer interrupt handler.

PAPI_overflow

- int PAPI_overflow(int EventSet, int EventCode, int threshold, int flags, PAPI_overflow_handler_t handler)
- Sets up an EventSet such that when it is PAPI_start()'d, it begins to register overflows
- The EventSet may contain multiple events, but only one may be an overflow trigger.

Statistical Profiling

- PAPI provides support for execution profiling based on any counter event.
- PAPI_profil() creates a histogram of overflow counts for a specified region of the application code.

PAPI_profil

int PAPI_profil(unsigned short *buf, unsigned int bufsiz, unsigned long offset, unsigned scale, int EventSet, int EventCode, int threshold, int flags)

- buf – buffer of bufsiz bytes in which the histogram counts are stored
- offset – start address of the region to be profiled
- scale – contraction factor that indicates how much smaller the histogram buffer is than the region to be profiled

PAPI 2.1 Release

- Platforms
  - Linux/x86, Windows 2000
    - Requires patch to Linux kernel, driver for Windows
  - Linux/IA-64
  - Sun Solaris/Ultra 2.8
  - IBM AIX/Power
    - Requires pmtoolkit (available from http://alphaworks.ibm.com/)b
  - SGI IRIX/MIPS
  - Cray T3E/Unicos
- Fortran and C binding and MATLAB wrappers

Performance Monitoring Modes

- Counting mode
  - Aggregate counts of event occurrences
  - Used to measure aggregate performance characteristics of application or system under study
- Sampling mode
  - Statistical sampling based on counter overflows
  - Used to relate performance problems to program locations

Performance Monitoring Modes (cont.)

- Platforms vary in support
  - SGI IRIX on MIPS R10K/12K - both
  - IBM Power 3 – counting mode
  - Compaq Alpha – sampling mode
  - IA-64 – both
- Either mode derivable from the other

Example: SGI IRIX

- perfex
  - Used to run a program and report “exact” counts of any two selected events for R10K/12K hardware counters, or to time-multiplex all 32 countable events
- ssrun
  - Run program in sampling mode to determine where in program events are occurring

Example: Compaq DCPI

- Compaq (formerly Digital) Continuous Profiling Interface
- Implemented on Alpha 21264A and later processors
- Based on instruction sampling with random period
- Interrupts running program, selects inflight instruction, and writes (exact) program counter and performance register values to database

DCPI (cont.)

- DCPI data collection is system-wide
- DCPI database is organized by epoch, process ID, and executable image
- Analytical tools
  - dcpiprof
  - dcpilist
  - dcpitopcounts

DCPI (cont.)

Examples:

- To display number of retired instruction samples and process cycle samples for each procedure:

  dcpiprof –event cycles –pm retired <image>

- For line-by-line display of ratio of retired instructions to replays:

  dcpilist –pm retired+retired::replays <procedure> <image>

- To display instructions with highest average retire delay:

  dcpitopcounts –pm valid:retdelay::valid <image>

Example: IA-64 PMU

- Supports access to counters in either counting or sampling mode
- Special features
  - Dedicated overflow interrupt mechanism
  - Data and instruction address and opcode qualification
  - Event Address Registers (EARs)
  - Branch Trace Buffer (BTB) event capture
- Access through **pfmon** and **pfmlib** by Stephane Eranian (Hewlett-Packard)

IA-64 PMU (cont.)

- Counting mode example:

  pfmon –e cpu_cycles,ia64_inst_retired <command>

- Sampling mode example:

pfmon –smpl-period=50000 –e cpu_cycles,ia64_inst_retired <command>

Sources of Error

- Perturbation by counter interfaces
  - Extra instructions
  - System calls
  - Cache pollution
  - Servicing interrupts
- PC sampling smear and skew on out-of-order processors
- Software multiplexing estimates counts from time-sliced values

Perturbation Errors

- Two types of errors discovered in microbenchmark studies
  - Constant bias
    - Can be compensated for
  - Variable error
    - Becomes insignificant if measured code is increased to sufficient granularity so that counter interface does not dominate

PC Sampling Error

- Non-constant bias
- Does not converge to expected value with greater number of samples
- DCPI solves this problem by knowing the precise program counter and execution history of randomly sampled instruction (supported in hardware).
- IA-64 provides a set of Event Address Registers (EARs) that capture where cache and TLB misses occur.
  - Randomized choice of which load instruction is traced

Statistical Sampling Error

- Sample value converges to expected value as the number of samples increases.
- Infrequent events or long sampling intervals require longer runs to get accurate estimates.
- Shorter sampling intervals increase sampling overhead.

Overhead of PAPI read

- Overhead in cycles per read call:

Linux/x86	Cray T3E	Linux/IA-64	IBM Power3	MIPS R12K
1299	1514	6526	3126	9810

Counting vs. Sampling Overheads

- EVH1 code instrumented with PAPI using TAU runs 5-10% slower
- Reports of 7-8x slowdown with PAPI instrumentation in tight loop on IBM Power3, 100x slowdown on SGI
- Sampling overhead using DCPI is usually around 0.05% but need at least 100 samples, or 6 million cycles, for accurate results.

Implications for PAPI

- Use hardware support for profiling where available
- Use programmable events to set up control blocks and return additional information

  PAPI_add_pevent

  PAPI_pread
- Standardize new features
  - Data and instruction address qualification
  - Opcode matching
  - Latency qualification/measurements

Implications for PAPI

- More experimental work needed to determine tradeoffs between accuracy and efficiency for counting vs. sampling mode on various platforms
  - New PAPI substrate for Compaq Alpha using new DCPI
  - Itanium

Implications for PAPI (cont.)

- Expand support for sampling mode in PAPI beyond current **PAPI_overflow**, **PAPI_profil**
- Handle choice between counting and sampling mode at tool level so that users can request performance data they want without worrying about how to do the instrumentation

Determining the Accuracy of Event Counts - Methodology

- Design and implement microbenchmarks for
  - Event count prediction
  - Determination of algorithm implemented in processor
    - E.g., branch prediction or cache prefetch algorithm
  - Predicted event count verification
- Collect data
  - Scripts permit a large number of experiments to be performed
  - Means and standard deviations computed
- Compare predicted and actual event counts
- Repeat process if necessary

Microbenchmarks

- Contain simple code segments
- Small when possible
- Comprised of regular patterns that permit mathematical modeling of associated event count
- Scalable w.r.t. granularity, i.e., number of generated events

Four Classes of Errors

- Overhead Bias
- Multiplicative
- Random
- Unknown
  - not predictable but verifiable
  - not predictable and cannot determine veracity
    - e.g., randomness in algorithm does not allow prediction

When Event Counts Can Be Used to Tune Performance

- Overhead Bias Error
  - adjust counts or granularity accordingly
- Multiplicative
  - adjust counts accordingly
- Random
  - perform multiple experiments and verify that standard deviation is small
- Unknown, not predictable but verifiable
  - not useful for fine performance tuning but useful for coarse tuning

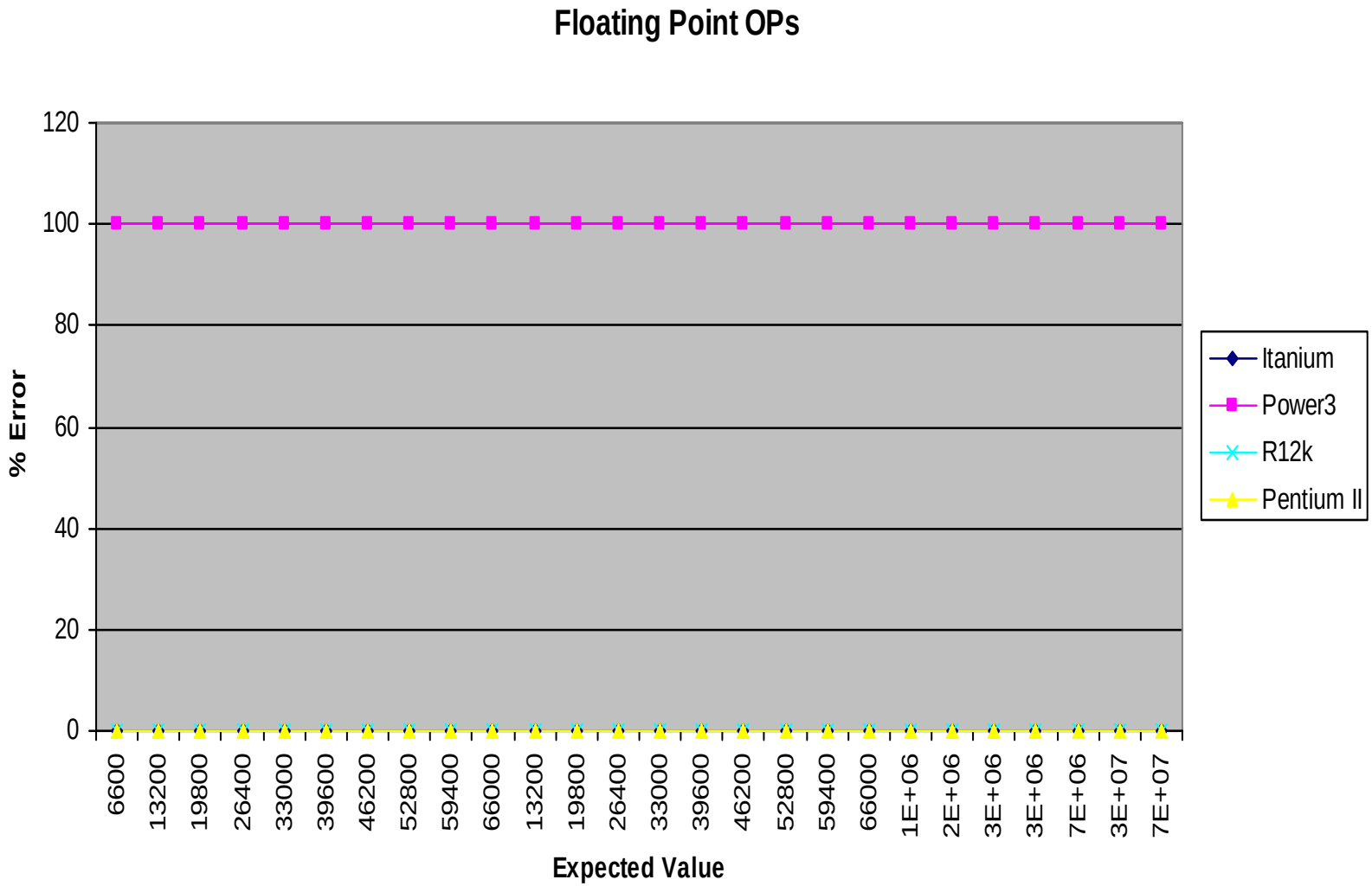
When Event Counts Cannot Be Used to Tune Performance

- Unknown
  - vendor assistance is needed to understand what is being counted or what algorithm is implemented in the processor
  - segregate combinations of error classes

Overhead Bias Error

	Itanium	**Power3**	**R12K**	**Pentium**
Loads	86	28	46	N/A
Stores	129	31	Mult. Error	N/A

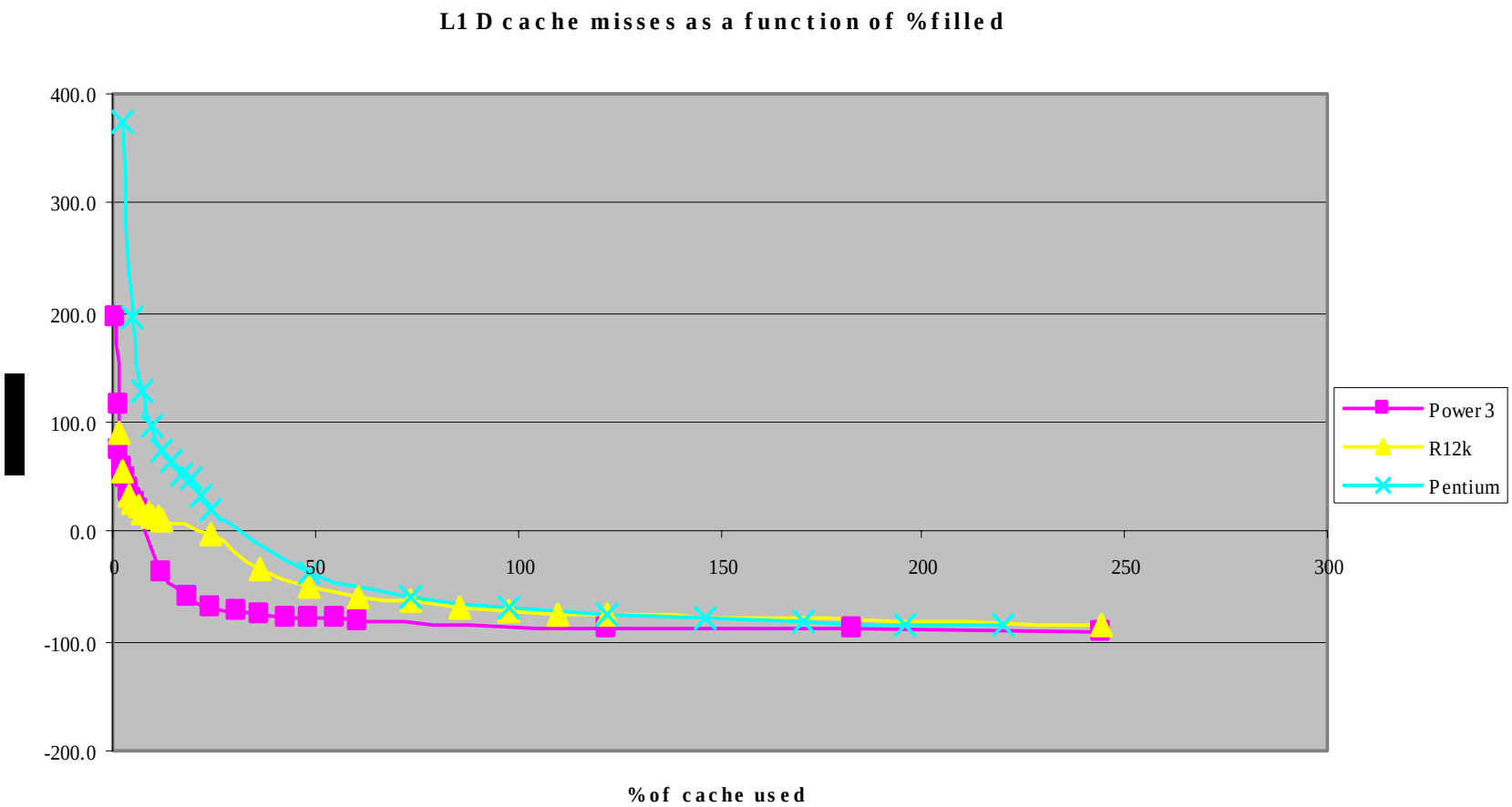
Multiplicative Error



Random Error

Itanium L1 Data Cache Misses	**Mean**	**Standard Deviation**
90% of data – 1M accesses	1,290	170
10% of data – 1M accesses	782,891	566,370

Unknown – Not Predictable But Verifiable



Unknown – Not Predictable and Not Verifiable

- Branch prediction
  - Algorithms used for prediction are very complex
  - Without proprietary information cannot make predictions

Future Work

- Expand events and platforms studied
- Compare accuracy of sampling with that of aggregate counts
- Determine usefulness of event counts generated by both sampling and aggregate counts for specific DoD applications